НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Департамент программной инженерии
Дисциплина: «Архитектура вычислительных систем»

Домашнее задание

Многопоточное вычисление обратной матрицы с использованием библиотеки OpenMP

ВАРИАНТ 4

Отчет

Листов 16

Выполнил:
Асатрян Эмин,
студент гр. БПИ198

Москва
2020

СОДЕРЖАНИЕ

1. ТЕКСТ ЗАДАНИЯ

Найти обратную матрицу для матрицы A. Входные данные: целое положительное число n, произвольная матрица A размерности n x n. Количество потоков является входным параметром, при этом размерность матриц может быть не кратна количеству потоков.

2. ОПИСАНИЕ ВХОДНЫХ ДАННЫХ

На вход программе подаются два аргумента через командную строку.

Первый аргумент – целое число, размерность матрицы $n \in [1, 10]$. Подобно ограничение было наложено из-за долгого ожидания выполнения задачи при больших значениях n.

Второй аргумент – целое число, кол-во создаваемых потоков $threadsCount \in [1, 1000]$.

3. ОПИСАНИЕ ВЫХОДНЫХ ДАННЫХ

В качестве результата программа выводит в командную строку сгенерированную целочисленную матрицу, а затем выводит информацию о подсчете обратной матрицы (если обратную матрицу посчитать возможно, то программа выведет эту матрицу в вещественных числах).

4. ОПИСАНИЕ ПРИМЕНЯЕМЫХ РАССЧЕТНЫХ МЕТОДОВ

4.1. Теория решения основной задачи

Для подсчета обратной матрицы была использована следующая формула

$$A^{-1} = \begin{pmatrix} \frac{A_{11}}{\Delta} & \frac{A_{21}}{\Delta} & \frac{A_{31}}{\Delta} \\ \frac{A_{12}}{\Delta} & \frac{A_{22}}{\Delta} & \frac{A_{32}}{\Delta} \\ \frac{A_{13}}{\Delta} & \frac{A_{23}}{\Delta} & \frac{A_{33}}{\Delta} \end{pmatrix},$$

где A_{ij} – алгебраические дополнения, а Δ – определитель матрицы A.

Для этого был использован принцип итеративного параллелизма, поскольку в общем случае, вычисление алгебраических дополнений каждого отдельного элемента занимает примерно равное количество времени. Был разработан метод fillInverseMatrix, вызываемый разными потоками, где каждый поток выполняет практически идентичную задачу – подсчет строк обратной матрицы. Таким образом, каждый поток будет заполнять определенные строки обратной матрицы, не конфликтуя с другими потоками. Каждому потоку, кроме последнего нужно будет заполнить по n / threadsCount строк обратной матрицы, а в случае, когда n не кратно threadsCount, последний поток возьмет на себя задачу заполнения оставшихся строк, причем их количество будет не больше threadsCount, поскольку оно равно n % threadsCount.

4.2. Методы

1) int** getSubMatrix(int** matrix, int i, int j, int n) - Метод получения подматрицы из матрицы, matrix - входная матрица, i - номер строки для удаления, j - номер столбца для удаления, n - размерность matrix

2) int determinant(int** matrix, int n) - Метод нахождения определителя матрицы, matrix - матрица, n – размерность

3) void display(int** matrix, int n) - Метод вывода целочисленной матрицы, matrix – матрица, n – размерность

4) void display(double** matrix, int n) - Метод вывода вещественной матрицы, matrix – матрица, n – размерность

5) int** generateRandomMatrix(int n) - Метод генерации случайной целочисленной матрицы, n – размерность

6) void fillInverseMatrix(int threadsCount, int threadInd, int n) - Метод параллельного заполнения обратной матрицы, threadsCount кол-во потоков, threadInd - индекс потока, выполняющего заполнение, n – размерность

7) void cleanMemory(int n) - Метод высвобождения памяти после успешного завершения программы, n - размерность созданных матриц

5. ТЕСТИРОВАНИЕ ПРОГРАММЫ

5.1. Корректные входные данные

1) Пример работы с одним потоком (рис. 1)

```
C:\Users\Emin\CLionProjects\InverseMatrixWithThreads>main 3 1
Generated matrix:
0       -3      0
4       4       1
-4      0       0
Inverse matrix:
0.000   0.000   -0.250
-0.333  0.000   0.000
1.333   1.000   1.000
```

Рисунок 1. Результат ввода n = 3, threadsCount = 1.

2) Пример работы с несколькими потоками (рис. 2)

```
C:\Users\Emin\CLionProjects\InverseMatrixWithThreads>main 3 3
Generated matrix:
-4      -5      0
-2      3       -4
-2      -3      5
Inverse matrix:
-0.029  -0.245  -0.196
-0.176  0.196   0.157
-0.118  0.020   0.216
```

Рисунок 2. Результат ввода n = 3, threadsCount = 3.

3) Пример работы с матрицей, размерность которой не кратна количеству потоков (рис. 3)

```
C:\Users\Emin\CLionProjects\InverseMatrixWithThreads>main 10 3
Generated matrix:
2       -3      0       2       3       3       1       1       2       2
4       0       0       -5      4       1       3       -2      -2      0
1       -2      0       -4      0       -2      -2      2       2       0
3       5       -5      -3      -3      -1      5       1       5       3
-1      5       -4      3       5       5       2       0       5       -3
4       -5      0       -2      1       -5      0       1       0       3
0       0       -2      1       5       -4      -4      0       5       2
5       -5      2       -3      -5      3       0       -1      4       -2
0       3       0       -3      3       5       4       1       0       -3
-3      1       5       3       -5      4       0       -1      5       0
Inverse matrix:
0.079   -1.386  -2.436  0.303   -1.944  -0.506  2.037   1.077   2.433   -0.869
0.017   -1.021  -1.790  0.257   -1.493  -0.466  1.559   0.726   1.851   -0.604
-0.023  -0.751  -1.343  0.110   -1.102  -0.157  1.142   0.523   1.453   -0.324
0.010   -0.573  -0.970  0.076   -0.621  -0.071  0.697   0.363   0.855   -0.288
0.012   0.042   -0.004  -0.041  0.027   0.050   0.045   -0.039  0.046   0.014
0.186   0.070   0.119   0.018   -0.036  -0.239  -0.047  0.000   -0.092  -0.029
-0.126  0.629   1.004   -0.132  1.002   0.513   -1.026  -0.507  -1.051  0.462
0.114   -1.025  -1.343  0.188   -1.216  -0.261  1.189   0.573   1.630   -0.538
-0.054  0.249   0.453   -0.044  0.419   0.179   -0.329  -0.165  -0.430  0.226
0.200   0.208   0.265   0.058   0.026   -0.145  -0.144  -0.161  -0.302  0.086
```

Рисунок 3. Результат ввода n = 10, threadsCount = 3.

4) Пример работы с потоками, количество которых превышает размерность матрицы (рис. 4)

```
C:\Users\Emin\CLionProjects\InverseMatrixWithThreads>main 5 111
Generated matrix:
3       3       -5      3       -3
-5      -5      -3      1       -1
-2      3       4       0       -2
3       0       0       0       -3
0       0       0       2       1
Inverse matrix:
-0.034  -0.082  -0.104  0.161   0.091
0.116   -0.079  0.085   -0.191  -0.134
-0.120  -0.023  0.082   0.137   0.192
0.017   0.041   0.052   0.086   0.454
-0.034  -0.082  -0.104  -0.173  0.091
```

Рисунок 4. Результат ввода n = 5, threadsCount = 111.

5) Обработка случая матрицы из одного элемента (рис. 5)

```
C:\Users\Emin\CLionProjects\InverseMatrixWithThreads>main 1 1
Generated matrix:
-1
Inverse matrix:
-1.000
```

Рисунок 5. Результат ввода n = 1, threadsCount = 1.

6) Обработка случая матрицы, определитель которой равен нулю (рис. 6)

```
C:\Users\Emin\CLionProjects\InverseMatrixWithThreads>main 2 2
Generated matrix:
1       -4
-1      4
Determinant of the generated matrix = 0, there is no inverse matrix.
```

Рисунок 6. Результат ввода n = 2, threadsCount = 2.

5.2. Некорректные входные данные

1) Обработка количества аргументов, меньшее чем 2 (рис. 7)

```
C:\Users\Emin\CLionProjects\InverseMatrixWithThreads>main
Incorrect amount of input arguments.
```

Рисунок 7. Результат запуска программы без аргументов.

2) Обработка количества аргументов, больших чем 2 (рис. 8)

```
C:\Users\Emin\CLionProjects\InverseMatrixWithThreads>main 1 1 1
Incorrect amount of input arguments.
```

Рисунок 8. Результат запуска программы с 3 аргументами.

3) Ввод строк в качестве аргументов (рис. 9)

```
C:\Users\Emin\CLionProjects\InverseMatrixWithThreads>main dssd dfdfs
Incorrect input arguments.
```

Рисунок 9. Результат запуска программы со строковыми аргументами.

4) Ввод размерности, меньшей нижней границы, равной 1 (рис. 10)

```
C:\Users\Emin\CLionProjects\InverseMatrixWithThreads>main 0 1
Incorrect input arguments.
```

Рисунок 10. Результат ввода n = 0, threadsCount = 1.

5) Ввод количества потоков, меньшего нижней границы, равной 1 (рис. 11)

```
C:\Users\Emin\CLionProjects\InverseMatrixWithThreads>main 1 0
Incorrect input arguments.
```

Рисунок 11. Результат ввода n = 1, threadsCount = 0.

6) Ввод размерности, большей верхней границы, равной 10 (рис. 12)

```
C:\Users\Emin\CLionProjects\InverseMatrixWithThreads>main 11 1
Incorrect input arguments.
```

Рисунок 12. Результат ввода n = 11, threadsCount = 1.

7) Ввод количества потоков, большего верхней границы, равной 1000 (рис. 13)

```
C:\Users\Emin\CLionProjects\InverseMatrixWithThreads>main 1 1001
Incorrect input arguments.
```

Рисунок 13. Результат ввода n = 1, threadsCount = 1001.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Архитектура параллельных вычислительных систем. Многопоточность [Электронный ресурс] – URL: http://softcraft.ru/edu/comparch/lect/07-parthread/ (дата обращения: 16.11.20).
2. Многопоточное программирование. OpenMP [Электронный ресурс] – URL: http://softcraft.ru/edu/comparch/practice/thread/03-openmp/ (дата обращения: 28.11.20).
3. Многопоточное программирование. Простые программы [Электронный ресурс] – URL: http://softcraft.ru/edu/comparch/practice/thread/01-simple/ (дата обращения: 16.11.20).
4. Поток выполнения [Электронный ресурс] – URL: https://ru.wikipedia.org/wiki/Поток_выполнения (дата обращения: 16.11.20).
5. Учебник по OpenMP – Блог программиста [Электронный ресурс] – URL: https://pro-prof.com/archives/4335 (дата обращения: 28.11.20).

ПРИЛОЖЕНИЕ 1. Код программы

```
#include <iostream>
#include <string>
#include <iomanip>
#include <omp.h>

using namespace std;

/**
 * Метод получения подматрицы из матрицы
 * @param matrix входная матрица
 * @param i номер строки для удаления
 * @param j номер столбца для удаления
 * @param n размерность matrix
 */
int** getSubMatrix(int** matrix, int i, int j, int n) {
    // Выделение памяти для подматрицы
    int** subMatrix = new int* [n - 1];
    for (int k = 0; k < n - 1; ++k) {
        subMatrix[k] = new int[n - 1];
    }

    int subMatrixRow = 0, subMatrixCol = 0;
    // Проход по всем элементам матрицы
    for (int matrixRow = 0; matrixRow < n; matrixRow++) {
        for (int matrixCol = 0; matrixCol < n; matrixCol++) {
            // Копирование тех элементов, которые не лежат в строке i и
столбце j matrix
            if (matrixRow != i && matrixCol != j) {
                subMatrix[subMatrixRow][subMatrixCol++] =
matrix[matrixRow][matrixCol];

                // Переход на новую строку в подматрице
                if (subMatrixCol == n - 1) {
                    subMatrixCol = 0;
                    subMatrixRow++;
                }
            }
        }
    }

    return subMatrix;
}

/**
 * Метод нахождения определителя матрицы
 * @param matrix матрица
 * @param n размерность
 */
int determinant(int** matrix, int n) {
    // Базовый случай
    if (n == 1)
        return matrix[0][0];

    int D = 0;

    // Подматрица matrix
    int** subMatrix;

    // Знак минора при разложении по первой строке
    int sign = 1;

    // Проход по первой строке
```

```
    for (int j = 0; j < n; j++) {
        subMatrix = getSubMatrix(matrix, 0, j, n);
        D += sign * matrix[0][j]
             * determinant(subMatrix, n - 1);

        // Смена знака и высвобождение памяти
        sign = -sign;
        for (int rowInd = 0; rowInd < n - 1; ++rowInd) {
            delete[] subMatrix[rowInd];
        }
        delete[] subMatrix;
    }

    return D;
}

/**
 * Метод вывода целочисленной матрицы
 * @param matrix матрица
 * @param n размерность
 */
void display(int** matrix, int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++)
            cout << matrix[i][j] << '\t';
        cout << endl;
    }
}

/**
 * Метод вывода вещественной матрицы
 * @param matrix матрица
 * @param n размерность
 */
void display(double** matrix, int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++)
            cout << matrix[i][j] << '\t';
        cout << endl;
    }
}

// Максимальное по модулю значение элементов матрицы
const int maxMatrixElementAbsValue = 5;

/**
 * Метод генерации случайной целочисленной матрицы
 * @param n размерность
 */
int** generateRandomMatrix(int n) {
    int** matrix = new int* [n];
    for (int i = 0; i < n; ++i) {
        matrix[i] = new int[n];
        for (int j = 0; j < n; ++j) {
            int randNum = rand() % (maxMatrixElementAbsValue + 1);
            matrix[i][j] = rand() % 2 ? randNum : -randNum;
        }
    }
    return matrix;
}

// Генерируемая матрица
int** matrix;
```

```
// Обратная матрица
double** inverseMatrix;

// Коэффициент перед транспонированной матрицей алгебраических дополнений
double multiplier;

/**
 * Метод параллельного заполнения обратной матрицы
 * @param threadsCount кол-во потоков
 * @param threadInd индекс потока, выполняющего заполнение
 * @param n размерность
 */
void fillInverseMatrix(int threadsCount, int threadInd, int n) {
    // Количество строк для заполнения каждым потоком, кроме последнего
    int linesOptimalPortion = n / threadsCount;

    // В случае, когда потоков более чем достаточно, игнорируются лишние
потоки, а остальные потоки
    // заполняют по одной строке
    if (linesOptimalPortion == 0) {
        linesOptimalPortion = 1;
        if (threadInd >= n)
            return;
    }

    // В условии цикла учитывается возможная потребность в заполнении
сверхнормативного кол-ва строк последним потоком
    for (int i = linesOptimalPortion * threadInd;
         i < linesOptimalPortion * (threadInd + 1) || (threadInd + 1 ==
threadsCount && i < n); ++i) {
        int sign = i % 2 ? -1 : 1;
        for (int j = 0; j < n; ++j) {
            int** subMatrix = getSubMatrix(matrix, i, j, n);
            inverseMatrix[j][i] = sign * determinant(subMatrix, n - 1) *
multiplier;

            // Смена знака и высвобождение памяти
            sign = -sign;
            for (int rowInd = 0; rowInd < n - 1; ++rowInd) {
                delete[] subMatrix[rowInd];
            }
            delete[] subMatrix;
        }
    }
}

/**
 * Метод высвобождения памяти после успешного завершения программы
 * @param n размерность созданных матриц
 */
void cleanMemory(int n) {
    for (int rowInd = 0; rowInd < n; ++rowInd) {
        delete[] matrix[rowInd];
    }
    delete[] matrix;
    for (int rowInd = 0; rowInd < n; ++rowInd) {
        delete[] inverseMatrix[rowInd];
    }
    delete[] inverseMatrix;
}

const int maxN = 10;
```

```
const int maxThreadsCount = 1000;

/**
 * @param argv argv[1] = n - размерность матрицы, argv[2] = threadCount -
кол-во потоков
 */
int main(int argc, char* argv[]) {
    // Добавление псевдослучайности и устанавливдение формата вывода
вещественных чисел
    srand(time(nullptr));
    cout << fixed << setprecision(3);

    // Проверка корректности аргументов командной строки
    if (argc != 3) {
        cout << "Incorrect amount of input arguments." << endl;
        return 1;
    }
    int n;
    int threadsCount;
    bool inputCorrectness = true;
    try {
        n = stoi(argv[1]);
        threadsCount = stoi(argv[2]);
    }
    catch (...) {
        inputCorrectness = false;
    }
    if (!(n >= 1 && n <= maxN && threadsCount >= 1 && threadsCount <=
maxThreadsCount && inputCorrectness)) {
        cout << "Incorrect input arguments." << endl;
        return 1;
    }

    // Генерация матрицы
    matrix = generateRandomMatrix(n);
    cout << "Generated matrix:" << endl;
    display(matrix, n);

    // Проверка обратимости матрицы
    int matrixDet = determinant(matrix, n);
    if (matrixDet == 0) {
        cout << "Determinant of the generated matrix = 0, there is no inverse
matrix." << endl;
        for (int rowInd = 0; rowInd < n; ++rowInd) {
            delete[] matrix[rowInd];
        }
        delete[] matrix;
        return 1;
    }

    // Выделение памяти для обратной матрицы
    inverseMatrix = new double* [n];
    for (int i = 0; i < n; ++i) {
        inverseMatrix[i] = new double[n];
    }

    // Подсчет коэффициента перед транспонированной матрицей алгебраических
дополнений и
    // подсчет обратной матрицы с помощью потоков (с обработкой случая n = 1)
    multiplier = 1 / (double)matrixDet;
    if (n > 1) {
#pragma omp parallel for
        for (int i = 0; i < threadsCount; ++i) {
```

```
            fillInverseMatrix(threadsCount, i, n);
        }
    }
    else {
        inverseMatrix[0][0] = multiplier;
    }

    // Вывод обратной матрицы и высвобожение памяти
    cout << "Inverse matrix:" << endl;
    display(inverseMatrix, n);
    cleanMemory(n);

    return 0;
}
```